Thinking Before Constraining: A Unified Decoding Framework for Large Language Models

Anonymous ACL submission

Abstract

Natural generation allows Language Models (LMs) to produce free-form responses with rich reasoning, but the lack of guaranteed structure makes outputs difficult to parse or verify. Structured generation, or constrained decoding, addresses this drawback by producing content in standardized formats such as JSON, ensuring consistency and guaranteed-parsable outputs, but it can inadvertently restrict the model's reasoning capabilities. In this work, we propose a simple approach that combines the advantages of both natural and structured generation. By allowing LLMs to reason freely until specific trigger tokens are generated, and then switching to structured generation, our method preserves the expressive power of natural language reasoning while ensuring the reliability of structured outputs. We further evaluate our approach on several datasets, covering both classification and reasoning tasks, to demonstrate its effectiveness, achieving a substantial gain of up to 27% in accuracy compared to natural generation, while requiring only a small overhead of 10–20 extra tokens. Our code and results are available online[1].

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide range of applications, including text completion, summarization, question answering, code generation, web navigation, data extraction, and tool use (Zhao et al., 2023; Minaee et al., 2024). Trained primarily on large natural language corpora, LLMs typically generate fluent and flexible text at inference time. As a consequence, they do not inherently guarantee adherence to predefined output structures. Although advanced LLMs often produce syntactically well-formed outputs, this behavior is not assured (Koo et al., 2024). The absence of strict structural guarantees can limit the applicability of LLMs in tasks such as schema-based information extraction, structured question answering, and many industrial use cases (Liu et al., 2024).

To address the need for structured output, several structured generation methods, such as Guidance (guidanceAI, 2025), Outlines (Willard and Louf, 2023), and XGrammar (Dong et al., 2024) have been proposed. These approaches guide language models to produce outputs that adhere to predefined formats or schemas, thereby improving reliability, parsability, and usability over downstream tasks. However, structured generation might constrain the expressiveness and naturalness of the model, which may reduce fluency and generalization (Tam et al., 2024; Banerjee et al., 2025).

In this work, we first introduce a simple approach that leverages both natural and structured generation. Here, structured generation primarily acts as a syntactic corrector and answer parser, and is only triggered when certain tokens are produced. We call this approach In-Writing. We then evaluate it on the downstream tasks, where prior studies have reported that structured generation methods underperform compared to alternative techniques (Tam et al., 2024), allowing us to reassess their effectiveness under our framework. Finally, we examine the performance of our approach on smaller models to determine whether lightweight architectures can also benefit from structured generation, while still maintaining strong overall performance.

Our main contributions are as follows:

- A simple yet effective approach that integrates natural generation with structured generation, enabling hybrid constrained decoding to function as both a syntactic corrector and a parser (see Figure 1).
- A comprehensive study of our approach on various datasets across different domains, including additional analyses on instruction opti-

[1] The code is appended as supplemental material. The repository will be made available upon acceptance.

Question: Alice, Bob, Claire, Dave, and Eve are dancers at a square dance. At the start of a song, they each have a partner: Alice is dancing with Helga, Bob is dancing with Ophelia, Claire is dancing with Sam, Dave is dancing with Melissa, and Eve is dancing with Lola. Throughout the song, the dancers often trade partners. First, Eve and Claire switch partners. Then, Bob and Eve switch partners. Then, Dave and Bob switch partners. Then, Alice and Bob switch partners. Finally, Alice and Claire switch partners. At the end of the dance, Bob is dancing with:
A) Helga. B) Ophelia. C) Sam. D) Melissa. E) Lola.
Answer: A

Natural Generation:
Let's break down the sequence of partner switches step by step:
1. Eve and Claire switch partners: Eve is now dancing with Sam, and Claire is dancing with Lola.
2. Bob and Eve switch partners: Bob is now dancing with Lola, and Eve is dancing with Melissa.
3. Dave and Bob switch partners: Dave is now dancing with Lola, and Bob is dancing with Melissa.
4. Alice and Bob switch partners: Alice is now dancing with Melissa, and Bob is dancing with Helga.
5. Alice and Claire switch partners: Alice is now dancing with Lola, and Claire is dancing with Helga.
At the end of the dance, Bob is dancing with... Helga!
✗

In-Writing:
Let's break down the sequence of partner switches step by step:
1. Eve and Claire switch partners: Eve is now dancing with Sam, and Claire is dancing with Lola.
2. Bob and Eve switch partners: Bob is now dancing with Lola, and Eve is dancing with Melissa.
3. Dave and Bob switch partners: Dave is now dancing with Lola, and Bob is dancing with Melissa.
4. Alice and Bob switch partners: Alice is now dancing with Melissa, and Bob is dancing with Helga.
5. Alice and Claire switch partners: Alice is now dancing with Lola, and Claire is dancing with Helga.
At the end of the dance, Bob is dancing with... Helga!
{ "final_answer": "A" }
✓

Figure 1: Comparison between Natural Generation and the proposed In-Writing method on Llama3-8B-it. Both produce identical reasoning; In-Writing additionally generates an easily parsable final answer with syntactic corrections (e.g., directing Helga to choice A).

mization and a review of previous work (Tam et al., 2024), demonstrating the benefits of this hybrid approach in enhancing LLM performance compared to natural generation.

- An investigation of structured generation on small models for deployment on edge devices.

This paper is structured as follows: In Section 2, we introduce preliminary work on Finite Automata. In Section 3, we explore related work on structured generation, discussing their advantages and some drawbacks. In Sections 4, 5, and 6, we present our method and experimental results to demonstrate its effectiveness and robustness.

2 Preliminaries

We first introduce the formal definition of a finite automaton, also known as a Finite-State Machine (FSM) (Sipser, 1996), which is one of the central components in constrained decoding (Willard and Louf, 2023; Koo et al., 2024), as they can represent a target regular expression (or regex)[2] and guide the model to select only tokens that are consistent with that regex at each decoding step.

Definition 1 (Finite Automaton)**.** *A finite automaton, or finite-state machine is a 5-tuple* $(Q, \Sigma, \delta, q_0, F)$*, where* Q *is a finite set of states,* Σ *a finite alphabet,* $\delta : Q \times \Sigma \to Q$ *the transition function,* $q_0 \in Q$ *the start state, and* $F \subseteq Q$ *the set of accept states.*

3 Related Work

Token generation in language models follows an autoregressive sampling procedure and can be represented by the conditional probability of the next token given all previous tokens (Bengio et al., 2003).

However, standard autoregressive sampling is inherently stochastic and does not guarantee any structured form in the generated output, which makes it difficult to apply directly in industrial applications where strict output formats are required (Scholak et al., 2021; Geng et al., 2023; Tam et al., 2024).

Tool calling, also referred to as function calling, was first introduced to enable LLMs to invoke external tools such as calculators, search engines, and Application Programming Interfaces (APIs) (Schick et al., 2023; Yao et al., 2022). Instructor

[2]We use the mathematical definition of regular expression (Sipser, 1996, Def. 1.52, p. 64)

(Liu and contributors, 2023) shows that by leveraging tool calling, LLMs can produce structured outputs such as JSON. However, since this mechanism is driven by the language model's predictions rather than hard decoding constraints, strict structural correctness is also not guaranteed.

A common approach to guarantee structured generation is to impose hard constraints on language models by masking the decoder logits during sampling (Zhang et al., 2019; Deutsch et al., 2019). The corresponding constrained decoding procedure is given in Algorithm 1.

Algorithm 1: LM Token Sampling with Masking

`sample_tokens_masked`(rx, p, L) ;
`//` rx`: regular expression,` p`: prompt,` L`: max new tokens`

1 $s \leftarrow (p)$; `// Generated token sequence`
2 **for** $i \leftarrow 1$ **to** L **do**
3     $\alpha \leftarrow LM(s, \theta)$;
4   Construct mask $m(s, rx)$;
5     $\tilde{\alpha} \leftarrow m \odot \alpha$;
6   Sample $\tilde{t}$ from $\tilde{\alpha}$;
7   **if** $\tilde{t} = EOS$ **then**
8   break;
9   **end**
10     $s \leftarrow \text{append}(s, \tilde{t})$;
11 **end**
12 **return** s;

Inspired by decoder logit masking, recent works have formulated regular expression–guided generation as an FSM (see Definition 1), which can be arbitrarily initialized and terminated (Willard and Louf, 2023; Koo et al., 2024).

Since regular languages can also be defined by regular expressions, which are equivalent to finite state automata (Koo et al., 2024; Kleene, 1956), it is possible to track the FSM states during the LLM token sampling process, as shown in Algorithm 1.

Recent works have empirically and theoretically observed that imposing constraints on LLM outputs can, in some cases, reduce performance accuracy for certain tasks (Tam et al., 2024; Banerjee et al., 2025). This raises the question of how to retain the benefits of constrained decoding while preserving the reasoning capabilities of LLMs.

Some approaches propose a two-step process, where a first model answers the question in a natural format, and a second model parses the final result, as in (Tam et al., 2024). These approaches require an additional, powerful model as a parser, which is costly and may introduce redundancy that could degrade overall performance (Lundberg and Ribeiro, 2023).

Other works attempt to combine free-form generation with structured generation (Banerjee et al., 2025) which focuses on context-free grammar cases. By introducing a start delimiter («) and an end delimiter (»), LMs can switch dynamically between constrained decoding inside the delimiters and free-form generation outside them. This hybrid strategy enforces syntactic validity where required while preserving the overall expressiveness of the generated text. However, it requires substantial preprocessing with fine-grained few-shot examples, as well as post-processing to parse the final output.

4 In-Writting Method

In this section, we detail our proposed approach, In-Writing, which combines both natural generation and structured generation. We focus on scenarios in which the desired outputs can be represented by regular expressions.

In our approach, summarized in Algorithm 2, the model first generates a reasoning trace without any constraints (preliminary state or state -1), which corresponds to lines 1–6. Once it produces a token belonging to a predefined set of trigger tokens, it switches to structured generation mode (state 0), which corresponds to lines 8–16. Thus, constrained decoding inherently functions as a parser that processes the final answer after the reasoning is completed.

Figure 2 illustrates this process using a simple regex that matches either "Yes." or "No.". In standard constrained decoding (Willard and Louf, 2023), generation starts directly in state 0 and is guided by an FSM that masks tokens inconsistent with the regex. In contrast, In-Writing defers constraint enforcement, allowing free-form reasoning in state -1 before switching to guided decoding once a trigger token is emitted, at which point only tokens compatible with the regex are allowed. Tokenization details may vary across language models, but the underlying mechanism remains unchanged.

The key advantages of this approach are as follows:

- **Single-model pipeline:** The same model both answers the question and parses the final output. It guarantees syntactical correctness,

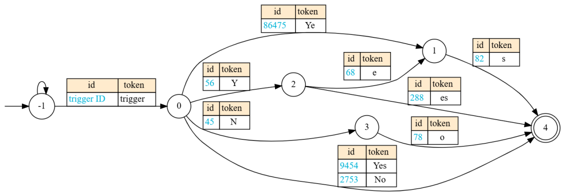

Figure 2: Illustrative example of the In-Writing approach. The model first generates unconstrained reasoning and then switches to guided decoding (state 0) once a trigger token is produced.

eliminating the need for a separate, larger parser model.

- **Leverages regular expressions:** If the set of possible answers is known in advance (e.g., numbers, letters, or multiple-choice options) and can be encoded in a regular expression, the method can enforce constrained decoding, ensuring that the model produces only valid outputs.

- **Robust final parsing:** The model can correctly parse the final result into predefined structured outputs without requiring sophisticated prompts, although well-designed prompts can further improve performance.

We name this approach In-Writing, inspired by **inpainting** technique in Diffusion Probabilistic Models (Lugmayr et al., 2022), where only a masked image region is regenerated using the surrounding context. Similarly, In-Writing allows the model to generate reasoning freely, then constrains only the relevant parts of the output to a predefined schema. By separating reasoning from formatting, In-Writing ensures expressive reasoning with structured outputs.

5 Experiments

This section presents the experimental evaluation conducted to demonstrate the practical feasibility of our approach. All experiments were performed on a single NVIDIA A40 GPU.

5.1 Datasets

Following Tam et al. (2024), we assess the efficiency and robustness of our hybrid approach on a diverse set of tasks, spanning from reasoning to classification, and covering a variety of output formats such as numerical values, textual strings, and multiple-choice responses. We adopt the same

Algorithm 2: LM Token Sampling with In-Writting Approach

sample_InWriting($rx, triggers, p, L$) ;
// rx: regular expression, $triggers$: trigger tokens, p: prompt, L: max new tokens

$s \leftarrow (p)$; // Generated token sequence
for $i \leftarrow 1$ **to** L **do**
  $\alpha \leftarrow LM(s, \theta)$;
  Sample t from α;
  if $t \notin$ *triggers* **then**
    $s \leftarrow$ append(s, t);
  end
  else
    while *rx is not completed* **do**
      Construct mask $m(s, rx)$;
      $\tilde{\alpha} \leftarrow m \odot \alpha$;
      Sample $\tilde{t}$ from $\tilde{\alpha}$;
      $s \leftarrow$ append$(s, \tilde{t})$;
    end
    return s;
  end
end
return s;

preprocessing procedures and data splits similar to their work.

5.1.1 Reasoning Tasks

GSM8K (Cobbe et al., 2021): A collection of grade-school math problems, designed to test the model's ability to generate intermediate reasoning steps.

Last Letter Concatenation (Wei et al., 2022): A task evaluating symbolic reasoning, where the model concatenates the last letters of a sequence of words into a string.

Shuffled Objects (Ghazal et al., 2013): A Big-Bench validation set in which the model predicts the final arrangement of objects after a sequence of shuffling actions.

5.1.2 Classification Tasks

DDXPlus (Fansi Tchango et al., 2022): A medical multiple-choice question task in which the model selects a diagnosis from 49 options. We also use the extracted subset provided by StreamBench (Wu et al., 2024).

MultiFin (Jørgensen et al., 2023): A classification task that assigns financial paragraphs to one of 5 predefined categories.

Sports Understanding (Ghazal et al., 2013): This task is derived from BigBench that tests whether short sports-related sentences are plausible or implausible.

NI - Task 280 (Mishra et al., 2022): A multiple-choice stereotype classification task based on a given paragraph, which is highly sensitive to the format of the prompt (Sclar et al., 2023).

5.2 Evaluation Metrics

We follow the same evaluation metrics as Tam et al. (2024). For answer generation tasks such as GSM8K and Last Letter Concatenation, we use exact match accuracy, which requires the model's output to exactly match the correct answer. For classification tasks such as Sports Understanding and DDXPlus, we use standard accuracy, comparing the model's predicted label to the correct label.

5.3 Prompts and Models

In the experiments of Tam et al. (2024), the same prompt template was used for every task, with the task description, format description, few-shot examples, and question text. For each task, there are 3 variations of the task description and 3 variations of the format description. The prompt template is provided in Appendix A.1.

For a direct comparison between our approach and the NL-to-Format (two-step) approach (Tam et al., 2024), we follow the same template and employ the same open-source models: LLaMA3-8B-Instruct (Dubey et al., 2024) and Gemma2-9B-Instruct (Team et al., 2024).

To further evaluate the capabilities of In-Writing on newer models and to explore strategies for optimizing prompts suitable for this hybrid approach, we conduct two additional, separate experiments:

1. **Instruction Format (In-Writing-IF):** We adopt a standard instruction format consisting of system, user, and assistant prompts, while keeping the prompt content largely unchanged.

   Unlike Tam et al. (2024), which places everything directly in a single user prompt, we present the examples through a multi-turn conversational flow between the user and assistant. The full prompt template is provided in Appendix A.1.

2. **Better Format (In-Writing-BF):** Building on the proper instruction format, we make minor adjustments to the syntax of the format descriptions to better align them with the desired output structure, as detailed in Appendix A.1.

For these experiments, we employ LLaMA3-8B-Instruct and a model from the latest Qwen family, Qwen3-8B (Yang et al., 2025), as these models support the full instruct format with system, user, and assistant roles.

Additionally, we re-evaluate the NL-to-Format and In-Writing approaches using the small language models to investigate whether smaller models can serve as an effective parser or whether a larger model is necessary for parsing the final results. This experiment aims to assess the feasibility of using our approach on edge devices. We use Qwen3-1.7B and SmolLM2 (Allal et al., 2025), both reported as top-tier small language models, as the primary models. Qwen3-32B, 4-bit quantized, serves as the parser for the NL-to-Format approach, while only the two small models are used in the In-Writing approach.

5.4 Output Schema

Although regular expressions can be specified independently, we use Pydantic[3] to automatically create and customize JSON schemas. The key used to extract the answer is always `final_answer`. Our JSON schema is defined as follows:

```
Class {task_name} (BaseModel):
    final_answer: {type, choice
     options, etc.} {optional pattern,
     if any}
```

While our approach is not limited to JSON, in this work we focus exclusively on JSON schemas due to the ease of parsing JSON into other formats such as XML or YAML.

5.5 Implementation Details

In-Writing can be implemented by modifying existing open-source libraries that support constrained decoding, such as Outlines (Willard and Louf, 2023). In our experiments, we use Litelines (Silva, 2025), which is also an open-source library based on Outlines, but with the `allow_preamble` feature. This feature allows LLMs to reason freely outside the schema until a trigger token (e.g., <eos>, {, etc.) is generated.

[3] https://docs.pydantic.dev/latest/

6 Results and Discussion

To assess the robustness of LMs in parsing answers, we report results averaged across all prompt variations, following the evaluation used in previous work (Tam et al., 2024).

6.1 Structured Generation Is an Effective Parser

We first evaluate our approach, In-Writing (denoted as **In-Writing-Base**), alongside the NL-to-Format approach (referred to as *Text* in prior work (Tam et al., 2024)), on the same 7 datasets, using the prompts and model configurations reported in the code released by Tam et al. (2024).

Although the rebuttal of (dottxt, 2025) raises concerns about the quality of these prompts, as they are arguably not comprehensive enough and contain some errors (see Appendix A.2), we reused them regardless, in order to enable a direct comparison between In-Writing and the NL-to-Format approach. This setup places our approach at a disadvantage, as there is no indication of which part of the output corresponds to the `final_answer` key in the defined schema.

The results reported in Tam et al. (2024) are limited to zero-shot prompting, even though few-shot experiments were also conducted. For a direct comparison, we follow the same zero-shot scenario. The results for the Reasoning and Classification tasks are presented in Table 1 and Table 2 respectively.

We observe that, even without explicit guidance regarding the `final_answer` key, In-Writing is still able to correctly parse the answer into the corresponding field and outperforms the NL-to-Format approach on almost every task (see Tables 1 and 2). However, In-Writing is not a perfect parser, and there are cases where the models fail to map the answer to the predefined schema. Some illustrative examples are provided in Appendix A.4.1.

While the NL-to-Format approach uses a larger model (*claude-3-haiku-20240307*), which is not open-source and thus cannot be reproduced, and is referred to as a "Perfect Parser" for extracting answers from the primary model's output, we found that the quality of the parsed answers varies tremendously. In Last Letter Concatenation, the parser performs extensive post-processing to retrieve the final answer. In contrast, for tasks such as DDXPlus, the parser fails to capture the final answer, resulting in significantly lower performance. An analysis is provided in Appendix A.4.2.

Model	**Task**	**NL-to Format**	**In-Writing -Base**
LLaMA	GSM8K	74.7 ± 0.6	**77.9** ± 1.3
Gemma	GSM8K	86.5 ± 0.6	**86.9** ± 0.4
LLaMA	Last Letter	**70.1** ± 5.3	69.1 ± 6
Gemma	Last Letter	56.8 ± 9.8	**65.9** ± 5
LLaMA	ShuffleObj	27.0 ± 5.5	**39** ± 6.7
Gemma	ShuffleObj	49.4 ± 5.8	**50.8** ± 4.7

Table 1: Zero-shot prompting results for LLaMA3-8B-it and Gemma2-9B-it, averaged over 9 variations of each reasoning task. The numbers indicate mean accuracy with standard deviation across variations.

Model	**Task**	**NL-to Format**	**In-Writing -Base**
LLaMA	MultiFin	60.3 ± 1.4	**64.5** ± 3
Gemma	MultiFin	70 ± 0.4	**86.9** ± 0.4
LLaMA	Sports	69.5 ± 12.7	**77.4** ± 4.3
Gemma	Sports	76.1 ± 2.3	**76.2** ± 1.8
LLaMA	Task280	65.3 ± 3.4	**74** ± 5.4
Gemma	Task280	69.8 ± 7.7	**74.5** ± 7.4
LLaMA	DDXPlus	12 ± 15.2	**30** ± 2.9
Gemma	DDXPlus	22.9 ± 5.8	**50.1** ± 1.6

Table 2: Zero-shot prompting results for LLaMA3-8B-it and Gemma2-9B-it, averaged over 9 variations of each classification task. The numbers indicate mean accuracy with standard deviation across variations.

6.2 Impact of Formats on Structured Generation Performance

Even though the models are often capable of identifying and correctly parsing the final result into the `final_answer` field, as shown in Tables 1 and 2, we hypothesize that their performance can be further improved through more effective prompt design. Specifically, providing (i) an improved instruction format (**In-Writing-IF**), (ii) a better output-aligned format built on top of the instruction format (**In-Writing-BF**).

We evaluate these approaches on 3 reasoning-task datasets only, both because Tam et al. (2024) have shown that structured generation struggles on these tasks and due to computational constraints.

Since no few-shot examples are provided in Tam et al. (2024) for the Shuffled Objects task, we adopt the few-shot example structure from the rebuttal work of dottxt (2025), using the first 4 samples of the Shuffle Object train dataset and the same answer format (see Appendix A.2).

Table 3 presents the results, showing that both In-Writing-IF and In-Writing-BF consistently improve the performance over In-Writing-Base, while also significantly outperforming Text-to-NL in the few-shot setting, except in the case of Qwen3-8B on GSM8K. The underlying reason is that when we explicitly instruct the model to produce a structured output format (e.g., `{"final_answer": <final answer>}`), in some GSM8K examples the model generates the trigger token (the opening brace, `{`), which immediately pushes it to produce the final answer prematurely due to the predefined schema. A similar phenomenon has been reported in prior work (Tam et al., 2024). We discuss solutions to mitigate this issue in Section 7.

Notably, In-Writing-IF reaches 80% accuracy and In-Writing-BF achieves 80.5% on GSM8K, both surpassing the reported 79.6% of LLaMA3-8B-it (8-shot CoT) (Dubey et al., 2024)

6.3 Experiments on Small Language Models

Some key advantages of structured generation are that it provides consistent, parsable outputs without requiring a larger model as a parser, extensive post-processing, or human intervention. These advantages naturally raise the question of whether smaller models can also benefit from structured outputs.

To answer this question, we evaluate In-Writing, as additional experiments, with Qwen3-1.7B and SmolLM2-1.7B-Instruct on 3 reasoning-task datasets, for the same reasons as in Section 6.2. We also reevaluate the NL-to-Format approach, using Qwen3-1.7B and SmolLM2-1.7B-Instruct to generate answers and Qwen3-32B (4-bit quantized) to parse the outputs of the primary model. The results of these experiments are reproducible and summarized in Table 4.

We observe that, on the LastLetter and ShuffleObj tasks, hybrid constrained decoding outperforms Text-to-NL, whereas on GSM8K, Text-to-NL substantially outperforms our method. This difference can be attributed to the fact that smaller models may inadvertently generate certain trigger tokens, such as `{`, during intermediate calculations, which immediately forces the model to follow the predefined schema. Similar phenomena were also observed in previous experiments (see Table 3). We discuss solutions to mitigate this issue in Section 7.

SmolLM2 failed on the Last Letter Concatenation task. This may be due to the fact that the task is particularly challenging for token-based models, which process tokens rather than individual characters, unlike humans. This observation, together with findings from previous work (Hsieh et al., 2023; Zhang et al., 2025), suggests that achieving strong performance on a specific task requires the model to be trained on that type of task in advance.

6.4 Tokens and Execution Time

While In-Writing shows very promising results (see Tables 1, 2, and 3), consistently outperforming the traditional two-step approach across all selected datasets, an important question is its impact on the number of additional tokens and execution time per sample.

Since our approach theoretically follows the same reasoning path as natural language generation, except when the trigger token is generated prematurely, the difference is limited to the final answer generation within the predefined schema.

We compare natural language generation, corresponding to the raw model output (i.e., the first step of the NL-to-Format pipeline and denoted as **NL**), with **In-Writing-Base**. These experiments use exactly the same prompt and role (user), ensuring a fair comparison.

We report the number of generated tokens and the execution time for 3 Reasoning tasks in Appendix A.5, in Tables 6, 7, and 8.

We observe that LMs require 10–20 additional tokens, compared to natural language generation, to produce the final answer within the predefined schema, resulting in an additional processing time of about 0.5–1.5 seconds, depending on the model and tokenizer. This additional time could be further reduced, as Litelines has not yet been fully optimized.

In contrast, when models generate the trigger token too early, which leads to degraded performance, as observed on the GSM8K task for small models (see Table 4), the number of generated tokens can be even lower than that of natural language generation. This observation highlights an additional practical use case for constrained decoding: when only the final answer is required, without intermediate reasoning (e.g., in data extraction), constrained decoding can reduce both the number of generated tokens and execution time, as it forces the model to generate the answer immediately.

Model	Task	Shot	NL-to Format	In -Writing -Base	In Writing -IF	In Writing -BF
LLaMA3-8B	GSM8K	0	74.7 ± 0.6	77.9 ± 1.3	79.2 ± 0.9	**79.9** ± 1
LLaMA3-8B	GSM8K	1	1.1 ± 0.4	3.1 ± 1	80 ± 0.6	**80.4** ± 0.7
LLaMA3-8B	GSM8K	4	X	24.3 ± 7.4	79.6 ± 0.3	**80.5** ± 0.6
Qwen3-8B	GSM8K	0	X	**79.1** ± 5.8	78.4 ± 4.9	76.5 ± 6.3
Qwen3-8B	GSM8K	1	X	**89.5** ± 0.5	85.6 ± 3	86 ± 3.1
Qwen3-8B	GSM8K	4	X	**89.4** ± 0.5	85.4 ± 2.4	86 ± 2.7
LLaMA3-8B	Last Letter	0	70.1 ± 5.3	69.1 ± 6	70.2 ± 6.6	**70.4** ± 7.9
LLaMA3-8B	Last Letter	1	44.6 ± 23.6	43.1 ± 38.6	77.9 ± 6.6	**81.9** ± 2
LLaMA3-8B	Last Letter	4	67.6 ± 3.9	77.4 ± 3.7	79.2 ± 2.9	**80.3** ± 1.6
Qwen3-8B	Last Letter	0	X	**74.7** ± 9.4	73.1 ± 7.1	74 ± 4.9
Qwen3-8B	Last Letter	1	X	70.8 ± 6.5	73.3 ± 4.4	**74.9** ± 2
Qwen3-8B	Last Letter	4	X	75.9 ± 1.9	77 ± 2.5	**78.7** ± 2.3
LLaMA3-8B	ShuffleObj	0	27 ± 5.5	**39** ± 6.7	34.8 ± 8.5	24.3 ± 10.3
LLaMA3-8B	ShuffleObj	1	X	22.4 ± 4.7	45.8 ± 3.1	**48.3** ± 1.3
LLaMA3-8B	ShuffleObj	4	X	38.2 ± 5.6	**46.7** ± 0.8	46.4 ± 0.6
Qwen3-8B	ShuffleObj	0	X	87.6 ± 2.7	88.7 ± 1.7	**89.9** ± 1.1
Qwen3-8B	ShuffleObj	1	X	82.6 ± 5	84.8 ± 2.9	**88.3** ± 1.6
Qwen3-8B	ShuffleObj	4	X	84 ± 3.7	86.7 ± 1.7	**88.7** ± 1.6

Table 3: Few-shot prompting results for LLaMA3-8B-it and Qwen3-8B, averaged over 9 variations of each reasoning task. Numbers indicate mean accuracy with standard deviation across variations. Boldface highlights the highest mean accuracy for each few-shot setting. ‘X‘ indicates that no information is available for that experiment.

7 Conclusion

We presented In-Writing, a simple yet effective approach that combines natural language generation with structured generation. Our experiments (see Tables 1, 2, 3, and 4) demonstrate that structured generation consistently improves model performance, particularly when the instruction prompt is aligned with the expected output format, or at least achieves comparable results (except in cases of premature trigger token generation). These gains are observed even for lightweight models, despite using the same experimental setup as Tam et al. (2024), which places our approach at a disadvantage. This contrasts with the observations of Tam et al. (2024). Importantly, these improvements are achieved with a reasonable increase in computational cost (see Tables 6, 7, and 8).

While constrained generation can perform well without sophisticated prompts and on general models, our results (see Tables 3 and 4) indicate that carefully designed prompts, well-formatted instructions, models fine-tuned for the specific task, and a well-chosen list of trigger tokens can prevent premature constrained decoding and further enhance performance in a complementary manner.

Limitations

As the combination of natural and structured generation, our approach inherits certain limitations from both paradigms:

- Models may prematurely generate trigger tokens that, similar to traditional structured generation, could inadvertently restrict the model’s reasoning capabilities (see Tables 3 and 4).
- Models may produce responses that, similar to natural generation, could loop indefinitely, resulting in unparsable outputs and wasted tokens.

The issue of premature trigger token generation can be mitigated by setting `<eos>` as a unique trigger token, particularly when the objective is to maximize accuracy. However, in some cases, the model may generate the `<answer>` first and then continue with a `<reasoning trace>`, in which case setting `<eos>` as a unique trigger token can lead to wasted tokens and unnecessary computation time.

The looping issue, which is more common in small LMs under natural generation, can be mitigated by imposing a maximum token budget for the natural generation phase (i.e., state -1), after which the model transitions to the trigger state 0 and switches to guided decoding.

References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, and 1 others. 2025. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. 2025. Crane: Reasoning with constrained llm generation. *arXiv preprint arXiv:2502.09061*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Daniel Deutsch, Shyam Upadhyay, and Dan Roth. 2019. A general-purpose algorithm for constrained sequential inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 482–492.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2024. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*.

dottxt. 2025. Say what you mean. https://blog.dottxt.ai/say-what-you-mean.html. Accessed: 2025-12-07.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. 2022. Ddxplus: A new dataset for automatic medical diagnosis. *Advances in neural information processing systems*, 35:31306–31318.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured nlp tasks without finetuning. *arXiv preprint arXiv:2305.13971*.

Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, pages 1197–1208.

guidanceAI. 2025. guidance: A guidance language for controlling large language models. GitHub repository.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017.

Rasmus Jørgensen, Oliver Brandt, Mareike Hartmann, Xiang Dai, Christian Igel, and Desmond Elliott. 2023. Multifin: A dataset for multilingual financial nlp. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 894–909.

Stephen Cole Kleene. 1956. *Representation of events in nerve nets and finite automata*, volume 34. Princeton University Press Princeton.

Terry Koo, Frederick Liu, and Luheng He. 2024. Automata-based constraints for language model decoding. *arXiv preprint arXiv:2407.08103*.

Jason Liu and contributors. 2023. Instructor: Structured outputs for large language models. https://github.com/instructor-ai/instructor. Accessed: 2025-12-06.

Michael Xieyang Liu, Frederick Liu, Alexander J Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J Cai. 2024. " we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. 2022. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11461–11471.

Scott Lundberg and Marco Tulio Ribeiro. 2023. The art of prompt design: Prompt boundaries and token healing. *Medium, may*.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196*.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324*.

Alonso Silva. 2025. Litelines. `https://github.com/alonsosilvaallende/litelines`. Accessed: 2025-12-07.

Michael Sipser. 1996. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024. Let me speak freely? a study on the impact of format restrictions on large language model performance. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1218–1236.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Brandon T Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Vivian Chen, and Hung-yi Lee. 2024. Streambench: Towards benchmarking continuous improvement of language agents. *Advances in Neural Information Processing Systems*, 37:107039–107063.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Hao Zhang, Richard Sproat, Axel H Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

Jianhua Zhang, Yi Gao, Ruyu Liu, Xu Cheng, Houxiang Zhang, and Shengyong Chen. 2025. Can students beyond the teacher? distilling knowledge from teacher's bias. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22434–22442.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, and 1 others. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).

A Appendix

A.1 Prompt Templates Used in Experiments

For a direct comparison between the work of Tam et al. (2024) and our approach, we use the exact same prompt template and content. The base prompt template, in which all content is provided under the user role, is as follows:

> **Follow the instruction to complete the task:** `{task_description}`
> **Instruct:** `{format_description}`
> **Few-shot examples:** `{few_shots}`
> **Question:** `{question}`

For the experiments aimed at finding a better prompt structure to enhance LLM performance (In-Writing-IF and In-Writing-BF), we split the content into three roles: system, user, and assistant. The prompt template is as follows:

> **"role"**: "system"
> **"content"**: **Follow the instruction to complete the task:**
> `{task_description}`
> **Instruct:** `{format_description}`
>
> **"role"**: "user"
> **"content"**: 1st few-shot question

"role": "assistant"
"content": 1st few-shot answer
⋮
"role": "user"
"content": n-th few-shot question
"role": "assistant"
"content": n-th few-shot answer

"role": "user"
"content": Main question

For In-Writing-IF, the prompt content is largely the same as in Tam et al. (2024), except that we remove the phrase `"Here are some examples:"` to accommodate the multi-turn conversational flow between the user and assistant.

For In-Writing-BF, we introduce minor modifications to the syntactic structure of the prompt to better align with the desired output format.

In the work of Tam et al. (2024), the format description typically specifies the final answer as:

> The final answer is <answer>

We modify this by introducing a keyword, `final_answer`, indicating that the model should store the final answer inside this key:

> The final answer is {"final_answer": <final answer>}

Similarly, the few-shot example answers are adjusted. For instance, one few-shot example from the LastLetter task is changed from:

> Answer: The last letter of “Elon” is “n”. The last letter of “Musk” is “k”. Concatenating them is “nk”. The answer is nk.

to:

> Answer: The last letter of “Elon” is “n”. The last letter of “Musk” is “k”. Concatenating them is “nk”. The answer is {"final_answer": "nk"} .

A.2 Prompt Inconsistencies in Previous Work

By investigating the outputs of the models provided in previous work (Tam et al., 2024), where constrained decoding was shown to harm LLM performance, we found several errors and inconsistencies:

In the Shuffle Object Task Description Variations, task variation 1 and task variation 2 include seven answer choices, but the prompt incorrectly instructs the model to choose from only four choices:

Task Description Variation 1:

> In this task, you are tasked to answer the following commonsense knowledge task. Read carefully each of the last questions and think step by step before answering. Make sure the answer only contains one of these four choices: A, B, C, D, E, F, G.

Task Description Variation 2:

> Read carefully each of the last questions and think step by step before answering. Make sure the answer only contains one of these four choices: A, B, C, D, E, F, G. In this task, you are tasked to answer the following commonsense knowledge task.

Format variations are used to instruct LLMs to reason first and then provide answers in a specific format. These variations are intended to be task-independent and should therefore remain consistent across tasks. However, in the previous work (Tam et al., 2024), the format variations are sometimes kept unchanged and sometimes vary across tasks, which raises concerns about the consistency of the evaluation.

Format Variations for GSM8K and Last Letter Concatenation

Variation 1

```
Instruct: Provide your output in the
    following text format:
Answer: <reasoning first>. The final
    answer is <answer>
```

Variation 2

```
Instruct: Provide your output in the
    following text format:
Step by step reasoning: ...
Answer: The final answer is ...
```

Variation 3

```
Instruct: Provide your output in the
    following text format:
Answer: <think step by step>. The final
    answer is <answer>
```

Format Variations for Shuffled Objects

Variation 1

```
Instruct: Now, take a deep breath and
    work on this problem step by step
to derive the most likely choice.
Provide your output in the following
    valid text format:
Answer: ...reasoning here... The answer
    is ...
```

Variation 2

```
Instruct: Provide your output in the
    following text format:
Step by step reasoning: ...
Answer: The final answer is ...
```

Variation 3

```
Instruct: Now, take a deep breath and
    work on this problem step by step
to derive the most likely answer.
Provide your output in the following
    valid text format:
Answer: [think step by step] The answer
    is [answer here]
```

Format Variations for DDXPlus

Variation 1

```
Instruct: Now, take a deep breath and
    work on this problem step by step
to derive the most likely diagnosis.
Provide your output in the following
    valid text format:
Answer: ...reasoning here... The answer
    is ...
```

Variation 2

```
Instruct: Now, take a deep breath and
    work on this problem step by step
to derive the most likely diagnosis.
Provide your output in the following
    valid text format:
Step by step reasoning: ...
Answer: ...
```

Variation 3

```
Instruct: Now, take a deep breath and
    work on this problem step by step
to derive the most likely diagnosis.
Provide your output in the following
    valid text format:
Answer: [think step by step] The answer
    is [answer here]
```

We also identify some missing or uninformative few-shot examples in the work of Tam et al. (2024). For instance, in the Shuffled Object task, few-shot examples are not aligned with the target dataset. Examples include:

- **Question**: "Many people live in Ethiopia. The people are very thin and good at distance running."
  Response: `answer = "race"`
- **Question**: "The Norwegian man was boring."
  Response: `answer = "race"`
- **Question**: "A tailor is sloppy in what he sews."
  Response: `answer = "profession"`
- **Question**: "Judy's male friend is very weak."
  Response: `answer = "gender"`
- **Question**: "The woman believed the Bible was full of truth."
  Response: `answer = "religion"`

Inspired by the work of dottxt (2025), we construct our few-shot using the first 4 samples from the Shuffled Object training dataset and adopt their answer format as follows.

1. **Question.** Alice, Bob, Claire, Dave, and Eve are dancers at a square dance. At the start of a song, they each have a partner: Alice is dancing with Patrick, Bob is dancing with Sam, Claire is dancing with Jamie, Dave is dancing with Lola, and Eve is dancing with Melissa.

   Throughout the song, the dancers often trade partners. First, Dave and Eve switch partners. Then, Dave and Alice switch partners. Then, Eve and Alice switch partners. Then, Claire and Bob switch partners. Finally, Dave and Alice switch partners. At the end of the dance, Alice is dancing with?

   A) Patrick
   B) Sam
   C) Jamie
   D) Lola
   E) Melissa

   Answer. Dave and Eve switch partners, so Dave's partner is now Melissa and Eve's partner is Lola. Then Dave and Alice switch partners, so Dave's partner is Patrick and Alice's partner is Melissa. Then Eve and Alice switch partners, so Eve's partner is Melissa and Alice's partner is Lola. Then Claire and Bob switch partners, so Claire's partner is Sam and Bob's partner is Jamie. Finally, Dave and Alice switch partners, so Dave's partner is Lola and Alice's partner is Patrick. Alice is dancing with Patrick (choice A).

2. **Question.** Alice, Bob, Claire, Dave, and Eve are dancers at a square dance. At the start of a song, they each have a partner: Alice is dancing with Ophelia, Bob is dancing with Jamie, Claire is dancing with Melissa, Dave is dancing with Rodrigo, and Eve is dancing with Patrick.

   Throughout the song, the dancers often trade partners. First, Claire and Bob switch partners. Then, Claire and Eve switch partners. Then, Claire and Bob switch partners. Then, Eve and Dave switch partners. Finally, Claire and Alice switch partners. At the end of the dance, Alice is dancing with?

   A) Ophelia
   B) Jamie
   C) Melissa
   D) Rodrigo
   E) Patrick

   Answer. Claire and Bob switch partners, so Claire's partner is Jamie and Bob's partner is Melissa. Then Claire and Eve switch partners, so Claire's partner is Patrick and Eve's partner is Jamie. Then Claire and Bob switch partners, so Claire's partner is Melissa and Bob's partner is Patrick. Then Eve and Dave switch partners, so Eve's partner is Rodrigo and Dave's partner is Jamie. Finally, Claire and Alice switch partners, so Claire's partner is Ophelia and Alice's partner is Melissa. Alice is dancing with Melissa (choice C).

3. **Question.** Alice, Bob, Claire, Dave, and Eve are dancers at a square dance. At the start of a song, they each have a partner: Alice is dancing with Jamie, Bob is dancing with Lola, Claire is dancing with Izzi, Dave is dancing with Rodrigo, and Eve is dancing with Ophelia.

   Throughout the song, the dancers often trade partners. First, Bob and Eve switch partners. Then, Alice and Bob switch partners. Then, Dave and Alice switch partners. Then, Dave and Claire switch partners. Finally, Bob and Claire switch partners. At the end of the dance, Claire is dancing with?

   A) Jamie
   B) Lola
   C) Izzi
   D) Rodrigo
   E) Ophelia

   Answer. Bob and Eve switch partners, so Bob's partner is Ophelia and Eve's partner is Lola. Then Alice and Bob switch partners, so Alice's partner is Ophelia and Bob's partner is Jamie. Then Dave and Alice switch partners, so Dave's partner is Ophelia and Alice's partner is Rodrigo. Then Dave and Claire switch partners, so Dave's partner is Izzi and Claire's partner is Ophelia. Finally, Bob and Claire switch partners, so Bob's partner is Ophelia and Claire's partner is Jamie. Claire is dancing with Jamie (choice A).

4. **Question.** Alice, Bob, Claire, Dave, and Eve are friends and avid readers who occasionally trade books. At the start of the semester, they

each buy one new book: Alice gets *Catch-22*, Bob gets *The Hound of the Baskervilles*, Claire gets *Frankenstein*, Dave gets *The Pearl*, and Eve gets *The Fellowship of the Ring*.

As the semester proceeds, they start trading books. First, Eve and Alice swap books. Then, Alice and Claire swap books. Then, Alice and Bob swap books. Then, Dave and Alice swap books. Finally, Dave and Claire swap books. At the end of the semester, Dave has?

A) Catch-22
B) The Hound of the Baskervilles
C) Frankenstein
D) The Pearl
E) The Fellowship of the Ring

Answer. Eve and Alice swap books, so Eve has Catch-22 and Alice has The Fellowship of the Ring. Then Alice and Claire swap books, so Alice has Frankenstein and Claire has The Fellowship of the Ring. Then Alice and Bob swap books, so Alice has The Hound of the Baskervilles and Bob has Frankenstein. Then Dave and Alice swap books, so Dave has The Hound of the Baskervilles and Alice has The Pearl. Finally, Dave and Claire swap books, so Dave has The Fellowship of the Ring and Claire has The Hound of the Baskervilles. Dave has The Fellowship of the Ring (choice E).

A.3 Results of Experiments on Small Language Models

The results of experiments on small language models are summarized in Table 4.

A.4 Semantic Parsability of Parsing Approaches

We analyze two different approaches: Constrained Decoding as a parser, and NL-to-Format, which uses another, larger LLM as a parser.

A.4.1 Constrained Decoding Is Not a Perfect Parser

By examining the experimental results of our approach, we observe that hybrid constrained decoding is far from perfect. Although this method shows promising results compared to the NL-to-Format baseline (i.e., a two-step approach in which an additional large language model is used as a parser), it still fails in a non-negligible number of cases.

We identified some examples from the GSM8K benchmark where constrained decoding fails to correctly parse the final answer into the `final_answer` field. Some of these examples are shown below.

```
"sample_id": 63,
"gold_answer": "25000",
"model_output": "Answer: For the first 20 years, Marcy's annual pension
       remains the same at $50000. After 20 years, she becomes entitled
       to 5% of the value of the pension per year, which is 5% of $50000
       = 0.05 x $50000 = $2,500. The final answer is $52,500.

{  "final_answer" :  "52"  }",
"parsed_answer": "52",

"sample_id": 125,
"gold_answer": "10",
"model_output": "Answer: Let's start by identifying the number of
       letters that Jennie has already stamped, which is 30. Since she
       puts stamps on one-third of the letters needing stamps, the
       number of letters needing stamps is three times the number of
       already-stamped letters. Therefore, the number of letters needing
       stamps is 3 x 30 = 90.

Since Jennie has a pile of 60 letters needing stamps, the number of
       letters already in the pile of already-stamped letters when
       Jennie began is 30 - (90 - 60) = 30 - 30 = 0. The final answer is
       0.

{  "final_answer" :  "30"  }",
"parsed_answer": "30"
```

While we cannot provide a full statistical analysis due to human resource limitations, we would like to emphasize that constrained decoding is far from a perfect parser, and future work is needed to improve its semantic correctness and parsability.

A.4.2 Limitations of LLMs as Parsers

One direct limitation of using a larger LLM as a parser, as suggested by Tam et al. (2024), is the high computational cost, which can be impractical. Indeed, if a larger model is available, it raises the question of why a smaller model would be used in the first place.

Another issue we observed when analyzing the logs from this work is that LLMs used as parsers sometimes "**correct**" the output of smaller models. For instance, in the Last Letter task, models often return a "**-**" or **whitespace** between letters. The parser may attempt to remove these characters, which incidentally leads to the correct prediction. In such cases, the parser performs more than simple parsing, it effectively modifies and corrects the output to produce syntactically valid results.

LLMs as parsers are also prompt-dependent and can be unpredictable. For example, in the logs of DDXPlus task, the parser sometimes generates natural conversational phrases such as *"the final answer that matches the given list of valid diagnoses is"* before outputting the diagnosis ****acute dystonic reactions****. While the diagnosis itself is correct, this additional text renders the output unparsable.

Model	Task	Shot	NL-to Format	In -Writing -Base	In Writing -IF	In Writing -BF
Qwen3-1.7B	GSM8K	0	**73.8** ± 1.3	59 ± 5.4	53.3 ± 4.8	52.8 ± 4.9
Qwen3-1.7B	GSM8K	1	**71** ± 2.8	65.9 ± 1.5	58.2 ± 3.2	60.2 ± 2.4
Qwen3-1.7B	GSM8K	4	**73.2** ± 1.9	69.4 ± 1.5	64.6 ± 1.9	67.6 ± 1.7
SmolLM2-1.7B	GSM8K	0	36.3 ± 7.3	36.6 ± 8.4	**41.1** ± 6	40.1 ± 4.3
SmolLM2-1.7B	GSM8K	1	**38.6** ± 2.6	38.4 ± 3.1	30.4 ± 2.8	27.4 ± 4.1
SmolLM2-1.7B	GSM8K	4	**39.8** ± 1.5	39.1 ± 2.4	34 ± 1.6	38.8 ± 1.1
Qwen3-1.7B	Last Letter	0	60.4 ± 3.9	69.7 ± 4.5	**70.5** ± 1.9	69.7 ± 3.6
Qwen3-1.7B	Last Letter	1	58.4 ± 3.3	**66.7** ± 3.5	64 ± 3.5	63.9 ± 2.7
Qwen3-1.7B	Last Letter	4	56.7 ± 4	**66.7** ± 3.5	55 ± 2.9	55 ± 2.3
SmolLM2-1.7B	Last Letter	0	0.7 ± 1	**1.1** ± 1.4	0.8 ± 1.4	0.7 ± 0.5
SmolLM2-1.7B	Last Letter	1	3 ± 2.5	5.6 ± 2.8	8.4 ± 2.9	**8.9** ± 2.7
SmolLM2-1.7B	Last Letter	4	3 ± 1.2	5 ± 2.3	**10.8** ± 0.5	9.5 ± 0.5
Qwen3-1.7B	ShuffleObj	0	44.1 ± 1.4	43.4 ± 1.4	**45.4** ± 2.7	**45.4** ± 2.7
Qwen3-1.7B	ShuffleObj	1	34 ± 6.4	33.7 ± 6.5	**48.1** ± 1	46.2 ± 1.7
Qwen3-1.7B	ShuffleObj	4	35.6 ± 6.7	35.3 ± 6.5	46.2 ± 2	**46.3** ± 1.6
SmolLM2-1.7B	ShuffleObj	0	**18.2** ± 0.8	**18.2** ± 0.8	18 ± 1	18 ± 1
SmolLM2-1.7B	ShuffleObj	1	17.9 ± 1.1	17.6 ± 1	18.4 ± 1	**18.8** ± 0.6
SmolLM2-1.7B	ShuffleObj	4	18.1 ± 1	16.4 ± 1.3	**19.7** ± 0.6	19.2 ± 0.5

Table 4: Few-shot prompting results for Qwen3-1.7B and SmolLM2-1.7B, averaged over 9 variations of each reasoning task. Numbers indicate mean accuracy with standard deviation across variations. Boldface highlights the highest mean accuracy for each few-shot setting.

For the Last Letter Concatenation task, we searched for the gold answer in the raw outputs of the primary models. For both tasks, we evaluated the parsed predictions (potentially modified by larger LLM parsers) against the reported results (Tam et al., 2024). We did not analyze raw outputs for DDXPlus, as they often re-list all possible diagnoses, making such analysis uninformative.

Table 5 shows that the parser substantially improves performance on the Last Letter Concatenation task, whereas for DDXPlus it fails to follow the instructions for extracting the final answer, leading to a significant drop in accuracy.

A.5 Token Usage and Execution Time

We report the number of generated tokens and the execution time per sample for the 3 Reasoning tasks in Tables 6, 7, and 8.

Model	Task	Reported Result	Gold Answer in Raw Output	Gold Answer in Prediction
LLaMA3-8B	LastLetter	70.1	46.8	70.1
LLaMA3-8B	DDXPlus	12	X	49.1

Table 5: Comparison of reported results, gold answers in raw outputs, and gold answers in predictions after parsing by larger LLMs. ‘X‘ indicates that no information is available for that experiment.

Model	Shot	Type	Token	Time (s)
Qwen3-8B	0	NL	166.8	6.8
Qwen3-8B	0	In-Writing-Base	165.6	7.4
LLaMA-3-8B-it	0	NL	157.2	4.6
LLaMA-3-8B-it	0	In-Writing-Base	178.8	6.2
Qwen3-8B	1	NL	120	4.8
Qwen3-8B	1	In-Writing-Base	139	6.2
LLaMA-3-8B-it	1	NL	37.2	0.8
LLaMA-3-8B-it	1	In-Writing-Base	50.6	1.8
Qwen3-8B	4	NL	123.2	5
Qwen3-8B	4	In-Writing-Base	141.9	6.4
LLaMA-3-8B-it	4	NL	77	2
LLaMA-3-8B-it	4	In-Writing-Base	97.3	3.5
Qwen3-1.7B	0	NL	182.6	5.7
Qwen3-1.7B	0	In-Writing-Base	167.6	5.7
SmolLM2-1.7B	0	NL	188.7	3.7
SmolLM2-1.7B	0	In-Writing-Base	205.2	4.4
Qwen3-1.7B	1	NL	150.1	4.6
Qwen3-1.7B	1	In-Writing-Base	152.7	5.3
SmolLM2-1.7B	1	NL	164.8	3.2
SmolLM2-1.7B	1	In-Writing-Base	181.5	3.9
Qwen3-1.7B	4	NL	153.1	4.7
Qwen3-1.7B	4	In-Writing-Base	157.7	5.4
SmolLM2-1.7B	4	NL	168.9	3.4
SmolLM2-1.7B	4	In-Writing-Base	185.6	4

Table 6: Inference statistics on the GSM8K task, reporting the number of newly generated tokens and execution time (in seconds) for different model sizes, comparing natural language generation and natural language generation with constrained decoding (In-Writing-Base).

Model	Shot	Type	Token	Time (s)
Qwen3-8B	0	NL	122.1	4.9
Qwen3-8B	0	In-Writing-Base	136.2	6.3
LLaMA-3-8B-it	0	NL	93	2.5
LLaMA-3-8B-it	0	In-Writing-Base	104.6	3.5
Qwen3-8B	1	NL	86.9	3.3
Qwen3-8B	1	In-Writing-Base	102.2	4.9
LLaMA-3-8B-it	1	NL	82.9	2.4
LLaMA-3-8B-it	1	In-Writing-Base	96.2	3.4
Qwen3-8B	4	NL	93	3.7
Qwen3-8B	4	In-Writing-Base	108.6	5.2
LLaMA-3-8B-it	4	NL	99.2	2.7
LLaMA-3-8B-it	4	In-Writing-Base	112.5	3.8
Qwen3-1.7B	0	NL	144.3	4.4
Qwen3-1.7B	0	In-Writing-Base	154.8	5.2
SmolLM2-1.7B	0	NL	101.5	1.7
SmolLM2-1.7B	0	In-Writing-Base	116.6	2.3
Qwen3-1.7B	1	NL	145.7	4.5
Qwen3-1.7B	1	In-Writing-Base	156.6	5.3
SmolLM2-1.7B	1	NL	103.8	1.8
SmolLM2-1.7B	1	In-Writing-Base	119.1	2.4
Qwen3-1.7B	4	NL	134.4	4.1
Qwen3-1.7B	4	In-Writing-Base	145.3	5
SmolLM2-1.7B	4	NL	92.3	1.5
SmolLM2-1.7B	4	In-Writing-Base	107.5	2.3

Table 7: Inference statistics on the Last Letter Concatenation task, reporting the number of newly generated tokens and execution time (in seconds) for different model sizes, comparing natural language generation and natural language generation with constrained decoding (In-Writing-Base).

Model	Shot	Type	Token	Time (s)
Qwen3-8B	0	NL	279.1	11.9
Qwen3-8B	0	In-Writing-Base	293.8	13.8
LLaMA-3-8B-it	0	NL	201.6	6.1
LLaMA-3-8B-it	0	In-Writing-Base	217.3	7.4
Qwen3-8B	1	NL	213.7	9
Qwen3-8B	1	In-Writing-Base	229.8	10.8
LLaMA-3-8B-it	1	NL	77.3	2.2
LLaMA-3-8B-it	1	In-Writing-Base	92.2	3.4
Qwen3-8B	4	NL	207	8.9
Qwen3-8B	4	In-Writing-Base	222.3	10.6
LLaMA-3-8B-it	4	NL	188	6
LLaMA-3-8B-it	4	In-Writing-Base	204.3	7.3
Qwen3-1.7B	0	NL	285.9	9
Qwen3-1.7B	0	In-Writing-Base	298.3	10.4
SmolLM2-1.7B	0	NL	155.8	3.1
SmolLM2-1.7B	0	In-Writing-Base	169.5	3.5
Qwen3-1.7B	1	NL	267.8	8.4
Qwen3-1.7B	1	In-Writing-Base	279.7	9.7
SmolLM2-1.7B	1	NL	152.8	3
SmolLM2-1.7B	1	In-Writing-Base	166.4	3.4
Qwen3-1.7B	4	NL	273.2	8.7
Qwen3-1.7B	4	In-Writing-Base	285.2	10
SmolLM2-1.7B	4	NL	204.9	4.2
SmolLM2-1.7B	4	In-Writing-Base	216.7	4.6

Table 8: Inference statistics on the ShuffleObj task, reporting the number of newly generated tokens and execution time (in seconds) for different model sizes, comparing natural language generation and natural language generation with constrained decoding (In-Writing-Base).